一种可配置的CNN协加速器的FPGA实现方法

蹇 强[1],张培勇[1],王雪洁[2]
(1.浙江大学信息与电子工程学院,浙江杭州 310027;2.浙江大学城市学院,浙江杭州 310015)

摘 要: 针对卷积神经网络中卷积运算复杂度高而导致计算时间过长的问题,本文提出了一种八级流水线结构的可配置CNN协加速器FPGA实现方法.通过在卷积运算控制器中嵌入池化采样控制器的复用手段使计算模块获得更多资源,利用mirror-tree结构来提高并行度,并采用Map算法来提高计算密度,同时加快了计算速度.实验结果表明,当精度为32位定点数/浮点数时,该实现方法的计算性能达到22.74GOPS.对比MAPLE加速器,计算密度提高283.3%,计算速度提高了224.9%,对比MCA(Memory-Centric Accelerator)加速器,计算密度提高了14.47%,计算速度提高了33.76%,当精度为8-16位定点数时,计算性能达到58.3GOPS,对比LBA(Layer-Based Accelerator)计算密度提高了8.5%.

关键词: 卷积神经网络;FPGA;嵌入式;卷积计算;并行算法
中图分类号: TN47 **文献标识码:** A **文章编号:** 0372-2112 (2019)07-1525-07
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2019.07.017

An FPGA Implementation Method for Configurable CNN Co-Accelerator

JIAN Qiang[1],ZHANG Pei-yong[1],WANG Xue-jie[2]
(1. *College of Information Science and Electronic Engineering,Zhejiang University,Hangzhou,Zhejiang* 310027,*China*;
2. *Zhejiang University City College,Zhejiang University,Hangzhou,Zhejiang* 310015,*China*)

Abstract: To solve the problem that the time consumption of convolutional neural network is too much,which is mostly caused by the high complexity of convolution operation,an FPGA implementation of a configurable CNN co-accelerator with eight-stage pipeline structure is proposed. By embedding the pooling controller in the convolution controller,the computational module is able to obtain more resources. Specially,a mirror-tree structure is designed to increase parallelism. Furthermore,to increase computational density and speed up calculation at the same time,the Map algorithm is implemented in this design. The experimental results show that the computing performance of this implementation reaches 22.74 GOPS on 32-bit fixed/float point. Compared with MAPLE accelerator,the computational density is increased by 283.3%,and the calculation speed is boosted by 224.9%. Compared with MCA(Memory-Centric Accelerator), the computational density is increased by 14.47%,and the calculation speed is boosted by 33.76%. With a precision range between 8-bit and 16-bit fixed point,the performance reaches 58.3GOPS,and the computational density is increased by 8.5% compared with LBA(Layer-Based Accelerator).

Key words: convolutional neural network;FPGA;embedded-system;convolution;parallel algorithm

1 引言

神经网络在过去的数年间,已经在很多领域获得了出色的成果,例如视频监控、机器人视觉、数据中心的图像搜索引擎、图像识别等[1~6].卷积神经网络是目前性能较好的深度学习算法之一[7],与传统算法相比,神

收稿日期:2018-05-30;修回日期:2018-12-19;责任编辑:李勇锋
基金项目:面向14纳米及以下工艺的亚皮秒精度信号片上测量关键技术研究(No.61474098);面向10纳米及以下工艺集成电路晶圆快速缺陷检测(No.61674129)

经网络算法能在敏感区域识别和图像分类等任务上能获得更高的准确率. 如今,在很多嵌入式应用中人工智能任务的需求在不断的增长[8,9],深度学习为智能硬件,无人机,可穿戴设备等嵌入式应用带来了新的发展机遇,但是在嵌入式系统中进行人工智能应用开发仍然存在技术难点. 虽然卷积神经网络通过稀疏连接、权重共享、池化采样等技术手段来降低运算量,但是由于复杂的应用场景以及神经网络自身结构的特点,其运算量依然十分巨大,绝大多数嵌入式设备的处理器难以满足性能要求. 目前,很多学者对神经网络的硬件加速方法进行了研究,这些方法采用 FPGA 平台来设计硬件加速核[10],通过并行计算方式来为深度学习提供加速服务,取得了一定效果[11~13]. 本文在已有的并行加速算法的基础上,提出了的八级流水线结构,该结构能够在卷积控制器中嵌入池化采样逻辑,可以达到复用卷积控制器和池化控制器的目的,使计算逻辑获得更多的可用资源,同时,对浮点运算引入 Map 和 Reduce 操作来提高局部带宽来加速计算. 实验结果表明,该设计有效的提高了计算密度,并加快了计算速度.

2 系统结构

系统的整体结构如图 1 所示,主要包括 Cortex-A9 处理器、DDR 主存设备、以及 CNN 协加速处理器. Cortex-A9 处理器作为 Master 利用 AXI4Lite 总线对 CNN 协加速器进行配置及控制,可控制 CNN 协加速器的输入分辨率、输入通道、输出通道、卷积核的大小、功能选择等. CNN 协加速器利用 AXI 总线对 DDR 进行访问来获取所需的特征图及其卷积滤波器参数,同时将算得的特征图更新到 DDR 中去.

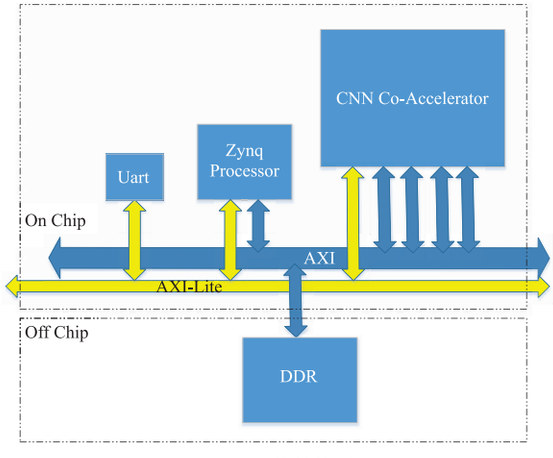

图1 系统结构图

协加速器的整体结构主要由三个部分组成,分别为片上储存、流处理器单元、以及之间的互联电路. 流处理器是卷积网络的基本处理单元,片上存储可提高访问数据的速度. 互联电路完成片上存储到流处理单元的逻辑映射关系.

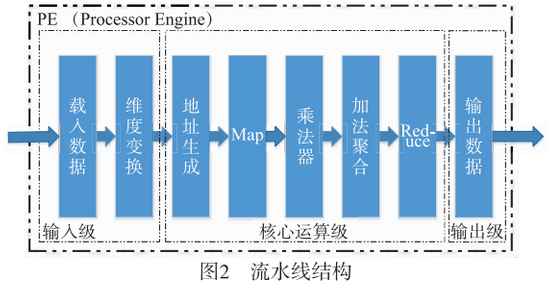

图2 流水线结构

CNN 的流水线结构如图 2 所示,由八级结构组成. 前两级结构负责将数据从 DDR 导入片上储存,并进行维度转换;中间五级完成整个卷积运算,其中,地址生成阶段将高维度矢量状态的数据转化为低维度数据流,并对数据流以卷积窗口为单位进行分割;乘法器阶段将图像信息与权重进行乘法操作;Map 和 Reduce 阶段在局部区域提高数据带宽;加法聚合阶段对窗口进行聚合操作;输出级将计算结果写回 DDR 中.

3 CNN 协加速器的实现方法

CNN 协加速器主要采用输入输出级优化、镜像树并行加速优化、池化控制器与卷积控制器复用优化和浮点运算优化四种优化策略设计实现. 约定卷积神经网络的参数如下,Tn 表示输入通道个数,Tm 表示输出通道个数,H 表示输入图像高度,W 表示输入图像宽度,K 表示卷积核的大小,Tr 表示输出图像高度,Tc 表示输出图像宽度.

3.1 输入输出级设计:

首先,鉴于访问 DDR 延迟会导致读写缓慢,可以通过加入 buffer 来改善这些不足. 同时,若将载入数据与计算分割为两个独立阶段,将会延长计算时间. 故将输入输出缓存设计为双 buffer 结构,buffer 分别有两种工作状态:装载状态和计算状态,将 buffer0 设置为装载状态,buffer1 设置为工作状态,此时 buffer0 将数据从 DDR 读入,buffer1 将数据提供给流处理器进行计算. 两个 buffer 的工作状态不断的切换,从而使装载时间和计算时间交叠来加快计算速度,如图 3 所示.

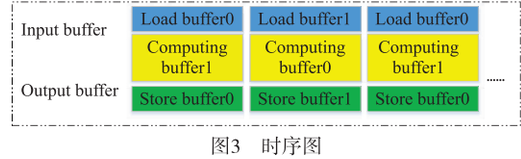

图3 时序图

3.2 流处理器并行加速

CNN 的卷积运算是主要是将每个输入通道的图像信息与权重系数按照固定的映射方式以窗口为单位进

行乘法求和运算. 算法的伪描述如图 4 所示.

```
for (int row = 0; row<H, row = row + S)
    for (int col =0; col<W; col = col + S)
        for (int out =0; out<Tm, out ++)
            for (int in=0; in<Tn, in++)
                for (int i=0; i<k; i++)
                    for (int j =0; j<k; j++){
                        output [out ][row / S][col / S] += \
                        (weight [out ][in][i][j] * \
                        image [in][row +i][col +j];
                    }
```

图4　伪代码描述

3.2.1　加法树

卷积神经网络通常计算量很大,以 AlexNet[14] 为例,计算总量为 1.33GFLOP,因此很多加速器通常采用加法树结构来进行计算(如图 5 所示). 加法树结构通过并行输入通道和输入通道来加快计算速度,但是会造成以下缺点.

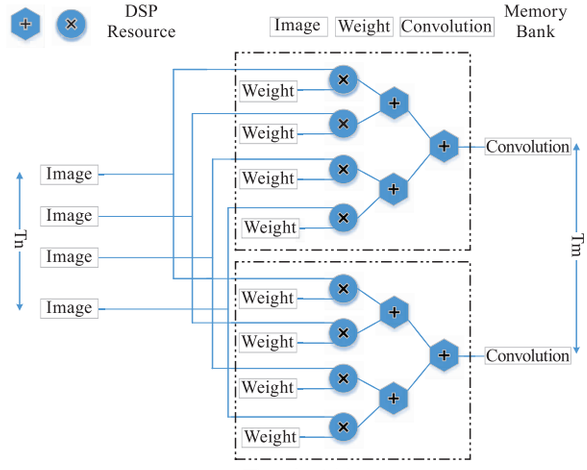

图5　加法树

(1)计算资源需求高

计算得一个加法树所需要的 DSP 资源如下:

$$DSP = Tm * (2 * Tn - 1) \tag{1}$$

分别对输出通道和输入通道求导得:

$$\begin{cases} \dfrac{\partial DSP}{\partial Tm} = 2 * Tn - 1 \\ \dfrac{\partial DSP}{\partial Tn} = 2 * Tm \end{cases} \tag{2}$$

首先,计算资源有限的 FPGA 设备无法同时满足多输入通道的和多输出通道的计算需求. 其次,当 Tm 和 Tn 很大的时候,DSP 随 Tm 和 Tn 的变化率很大,无法对 DSP 进行精细的调节,限制了并行度的提高.

(2)带宽需求高

数据传输的时间与卷积计算时间如下:

$$\begin{cases} \text{Communication time} = Tm * Tn * K * K + Tn * W * H \\ \qquad\qquad + Tm * Tr * Tc \\ \text{Computation time} = Tr * Tc * K * K \end{cases} \tag{3}$$

分析可知一般情况下,数据传输时间要大于卷积计算时间,这时带宽成为限制性能的主要因素,提高并行度并不会提高计算性能.

(3)可配置性差、资源利用率不高

加法树输入端口固定,难以适应输入通道发生变化的计算场景. 当输入通道小于加法树的输入端口个数时,DSP 的利用率将大幅度降低.

(4)采样电路需要重新设计

池化采样需要对各路输入分别进行采样,加法树则是对多路输入进行聚合,使控制器难以复用.

3.2.2　镜像树并行计算

流处理器分阶段进行卷积运算,地址生成器阶段将高维度的图像信息和权重信息按照卷积运算的映射方式进行降维流化;乘法器阶段将图像信息和权重信息进行相乘;加法阶段以流的基本单位进行聚合操作.

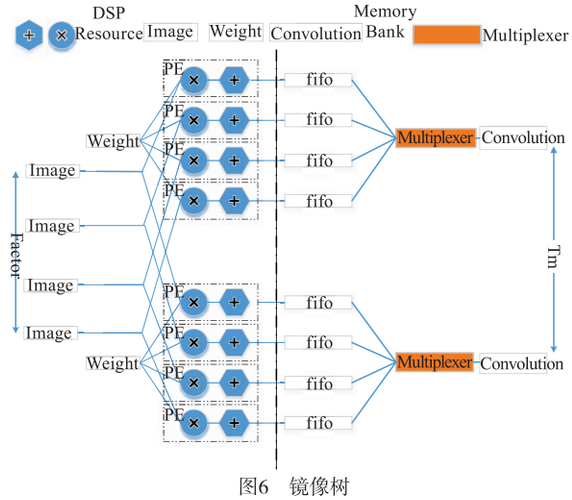

图6　镜像树

镜像树的结构如图 6 所示. 镜像树结构主要通过输入图像分割因子 Factor 和输出通道两个参数来提高乘法加法并行度.

首先,图像分割因子 Factor 是基于卷积神经网络权重共享的特点,这意味着同一帧图像的不同部分将共享相同的权重参数,进而可以对图像按照一定的分割因子进行分割提高并行度. 通过量化分析可知镜像树结构对 DSP 的资源消耗关系大致如下:

$$DSP = 2 * Tm * Factor \tag{4}$$

分别对输出通道和输入通道求导得:

$$\begin{cases} \dfrac{\partial DSP}{\partial Tm} = 2 * Factor \\ \dfrac{\partial DSP}{\partial Tn} = 0 \end{cases} \tag{5}$$

镜像树结构 DSP 随 Tn 的变化率为 0,随 Tm 的变化率为 2 * Factor,变化率小意味着可以对 DSP 进行精细的调节,相比加法树可以进一步提高并行度.

其次,镜像树结构的计算数据传输的时间与卷积计算时间如下:

$$\begin{cases}\text{Communication time} = Tm * K * K + W * H + Tm * Tr * Tc \\ \qquad\qquad \approx 1/Tn * \text{Old communication time} \\ \text{Computation time} = 1/\text{Factor} * (Tr * Tc * K * K \\ \qquad\qquad \approx 1/\text{Factor} * \text{Old computation time}\end{cases} \tag{6}$$

分析可知,镜像树结构的数据传输时间大约为加法树结构的 1/Tn,卷积计算时间为加法树结构的 1/Factor,通常 Tn 远大于 Factor,这意味着数据传输时间将不再大于卷积计算时间,带宽不再成为提高性能的瓶颈.

最后,由于镜像树结构并不在输入通道这个维度上提高并行度,流处理器将不再对输入通道参数敏感,这样即使输入通道变化时,所有的 DSP 都在进行有效计算,从而使计算效能接近性能峰值.

3.2.3 嵌入池化处理

池化采样和卷积操作基本一致,区别在于池化运算没有乘法阶段. 为了达到复用的目的,首先要解决这一运算过程中的异构性.

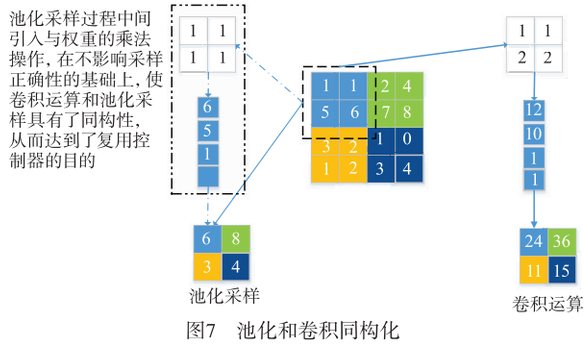

图7 池化和卷积同构化

对池化操作引入乘法运算,不会造成计算结果错误. 如图 7 所示,对于最大值池化操作,可以描述为对一个 2×2 的窗口进行最大值采样,也可以描述为将 2×2 的窗口与权重系数为 1 的核进行乘法运算后进行最大值采样. 因此,与权重的乘法操作使卷积运算和池化采样具有了同构性,卷积运算和池化运算可以复用相同的控制电路. 为了保证采样的正确性,池化采样在乘法阶段需要载入不同权重系数,对于最大值和最小值采样权重系数应设为 1,对于平均值采样权重系数应设为采样窗口大小的倒数.

其次,镜像树结构使得不同通道的输入图像被串行,不利于提高采样速度,对此采用图像空间和权重空间对调技术,通过将图像载入权重空间,将串行的图像并行化. 通过分析图 4 所示的伪代码可知. 未优化前 image 窗口和 weight 窗口通过同一个控制器生成,主要区

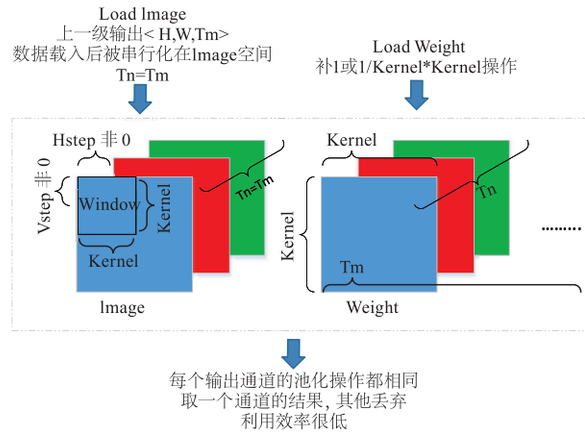

图8 未优化前的数据流

别在于 image 空间进行 VStep 寄存器(对应图 4 变量 row),HStep 寄存器(对应图 4 变量 col),和 Tn 寄存器(对应图 4 变量 in)偏置操作,在 weight 空间只进行 Tn 寄存器偏置操作,此时,图像信息被串行在 share memory 空间,视角如图 8 所示.

由于输出通道(对应图 4 变量 out 和变量 Tm)已经被分离,使得 image 和 weight 具有了对称特性,故对权重空间的地址生成同样引入 VStep 和 HStep 两个控制寄存器,在卷积运算时,将其值设为 0,对于控制器而言,image 空间和 weight 空间的地位完全相等,图像和权重的操作可以互换. 故对于池化操作,将 Tn 寄存器进行置 1 操作,image 空间的 VStep 寄存器和 HStep 寄存器进行置 0 操作,weight 空间的 VStep 寄存器和 HStep 寄存器进行偏置操作. Image 空间载入权重系数, Weight 空间载入图像信息,此时,池化采样被各个通道的流处理器并行运算,加快了运行速度. 视角如图 9 所示.

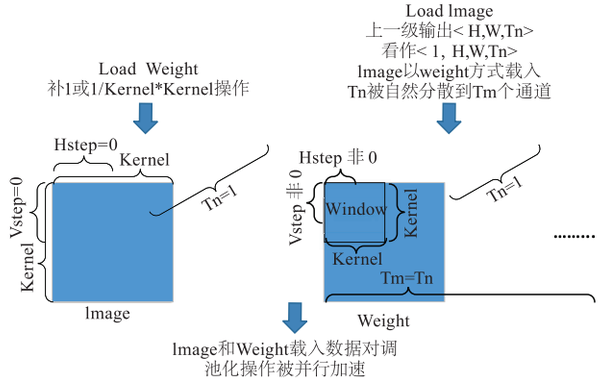

图9 优化后数据流视角

3.3 Map-Reduce 浮点数运算优化

对于浮点数运算而言,其计算无法在一个时钟周期里完成,因此聚合操作时每一步操作都要依赖于前一步的计算结果,使之难以进行流水线处理. 对此,采用

Map-Reduce 技术来进行加速. 通过在 Map 层进行分配,对数据进行 shuffle 操作,在有依赖的操作之间插入无依赖操作,使计算更为紧密,从而充分利用了加法器,有效的防止了流水线阻塞. 例如,如图 10 所示, op2 不依赖于 op1,op3 不依赖于 op2,故可在 op1 和 op4 之间,插入 op2 聚合 window2,插入 op3 聚合 window3,观察可知在进行 op4 时,其依赖的 op1 的结果已经被计算出来,从而提高计算密度,加快了计算速度. Reduce 操作则是 Map 操作的逆过程,负责对计算结果进行收集,并恢复数据的维度.

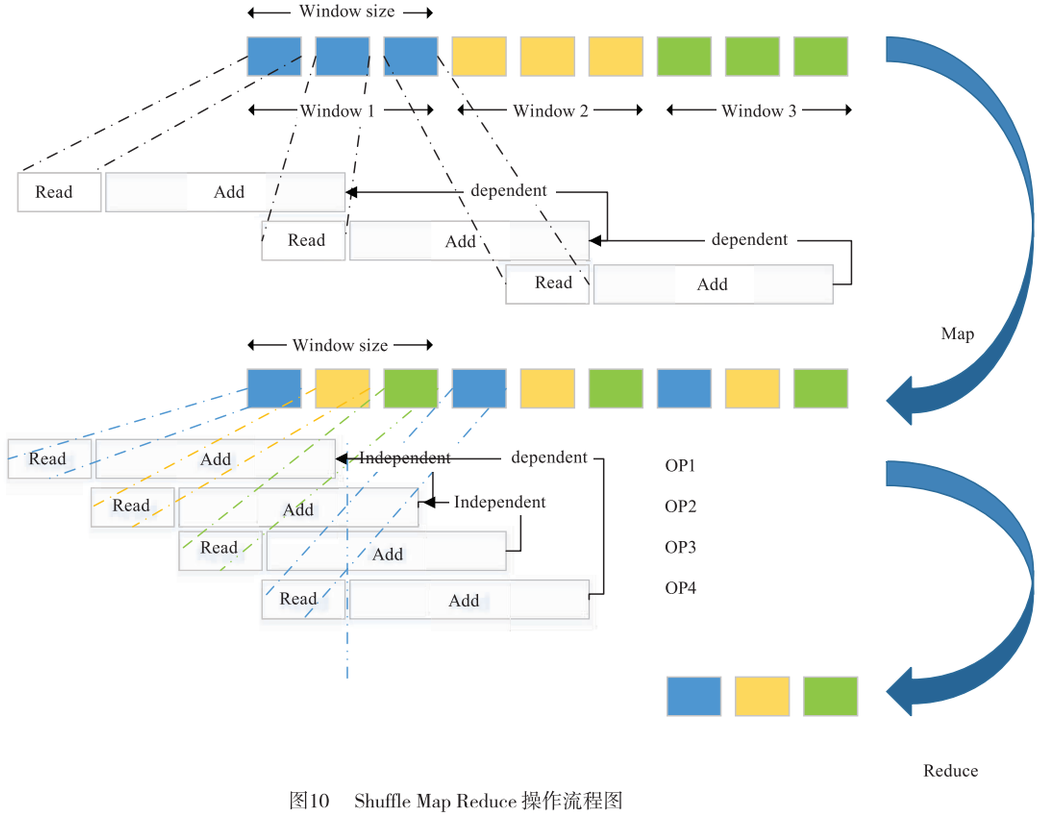

图10　Shuffle Map Reduce 操作流程图

4　实验结果与分析

本文采用流处理器并行加速的基本设计思想,以 Xilinx 公司生产的 ZC706 开发板为设计平台,最终完成了平台的设计与搭建. 经过综合后,资源消耗情况如表 1 ~2 所示.

表 1　加法树资源消耗[15]

输入通道:7		输出通道:64	
资源	已使用	资源	已使用
DSP	2240	2800	80%
BRAM	1024	2060	50%
LUT	186251	303600	61. 3%
FF	205704	607200	33. 87%

通过数据对比可知,在输出通道相同的情况下,采用流处理器设计后,DSP 的数量降为加法树的 26% ,FF 降为加法树的 40% ,LUT 降为加法树的 77% ,BRAM 基本持平.

表 2　流处理器资源消耗

输入通道:1 ~256		输出通道:64	
资源	已使用	可利用	占用率
DSP	596	900	66%
BRAM	1034	1090	94%
LUT	145257	218600	66%
FF	83948	437200	19%

由于加法树的输入通道是完全固定不变的,使得在输入通道小于 7 的计算情况下,须对空闲通道进行补 0 操作,大大降低了资源的使用效率,在输入通道为 1 时为最差情况,此时参与有效计算的 DSP 只有 320 个,利用率只有 14. 28% ,而流处理器可以将输入通道调节为 1,DSP 的使用率为 100% . 所以流处理器在资源利用率方面以及可配置性方面的表现远远要好于加法树.

流处理器的加速效果如表 3 所示. 分析可知该设计的资源利用率和计算密度获得了显著提高,对比

MAPLE 加速器和 MCA 加速器,计算密度提高了 14.47% 到 283.3%,计算速度提高了 33.76% 到 224.9%,对比 LBA[16]加速器,计算密度提高了 8.5%.

表 3 FPGA 平台对比

	PACT[11]	ISCA[12]	ICCD[13]	FPGA[16]	本文方法	
设备	SX240T	SX240T	VLX240T	VC709	Zc706	
容量	1056 DSP	1056 DSP	768 DSP	3600 DSP	900DSP	
频率	125MHz	200MHz	150MHz	110MHz	150MHz	100MHz
精度	16 Fixed	48 Fixed	Fixed	8 – 16 Fixed	32 Float	8 – 16 Fixed
运算量	0.53 GMAC	0.26 MAC	2.74 GMAC	0.78 GOP	1.33 GOP	1.74 GOP
性能	7 GOPS	16 GOPS	17 GOPS	213.70 GOPS	22.74 GOPS	58.3 GOPS
GOPS/DSP	0.66 E – 2	1.52 E – 2	2.21 E – 2	5.9 E – 2	2.53 E – 2	6.4 E – 2

表 4 不同平台比较

平台	CPU[16]	GPU[16]	SoC[16]	本文
型号	Intel 7700K	Nvidia GTX 980Ti	Snapdragon 821	Xilinx ZC706
性能 (GOPS)	15.4	365.11	4.4	58.3

将流处理器分别与不同平台的处理器进行对比,结果如表 4 所示. 流处理器的计算速度远远快于 CPU 设备,是 Intel 7700K 计算速度的 3.78 倍,是 Qualcomm Snapdragon 821 计算速度的 13.3 倍. 由于 GPU 的核心时钟大约在 1.2GHz ~ 1.4GHz 之间,FPGA 的核心频率只有 100MHz,且 GPU 并行处理器的个数要远大于 ZC706 开发板可部署的流处理器个数,故相比于 GPU,计算速度为其的 16%,但是在同频下,本文流处理器的计算效率更高. 综上所述,流处理器设计可以在嵌入式 CPU 计算能力不足的情况下,能够给嵌入式设备提供卷积运算能力.

5 结论

本文从提高资源利用率和提高计算速度的角度出发,通过合理的设计,提出了一种基于八级流水线结构的可配置 CNN 协加速器实现方法,通过复用控制器以及提高局部带宽的方式,有效的提高资源利用率,并加快了计算速度. 其计算速度是 Intel 7700K CPU 计算速度的 3.78 倍,SoC 计算速度的 13.3 倍. 对比 MAPLE 加速器和 MCA 加速器,计算密度提高了 14.47% 到 283.3%,计算速度提高了 33.76% 到 224.9%,对比 LBA 加速器,计算密度提高了 8.5%. 使得在嵌入式设备上进行 CNN 加速成为可能.

参考文献

[1] CFarabet, C Poulet, J Y Han, Y LeCun. CNP: An FPGA-based processor for convolutional networks[A]. 2009 International Conference on Field Programmable Logic and Applications[C]. Prague: IEEE, 2009. 32 – 37.

[2] Ji S, Xu W, Yang M, Yu K. 3D convolutional neural networks for human action recognition[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013, 35(1): 221 – 231.

[3] Larochelle H, Erhan D, Courville A, Bergstra J, Bengio Y. An empirical evaluation of deep architectures on problems with many factors of variation[A]. Proceedings of the 24th International Conference on Machine Learning[C]. New York: ACM, 2007. 473 – 480.

[4] Sankaradas M, et al. A massively parallel coprocessor for convolutional neural networks[A]. The 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors[C]. Boston: IEEE, 2009. 53 – 60.

[5] Bengio Y, Courville A, Vincent P. Representation learning: a review and new perspectives[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013, 35(8): 1798 – 1828.

[6] Yongmei Zhou, Jingfei Jiang. An FPGA-based accelerator implementation for deep convolutional neural networks [A]. The 4th International Conference on Computer Science and Network Technology (ICCSNT)[C]. China: IEEE, 2015. 829 – 832.

[7] Chen Y, et al. DaDianNao: a machine-learning supercomputer[A]. The 47th Annual IEEE/ACM International Symposium on Microarchitecture[C]. Cambridge: IEEE, 2014. 609 – 622.

[8] Roux S, Mamlet F, Carcia C. Embedded convolutional face finder[A]. 2006 IEEE International Conference on Multimedia and Expo[C]. Canada: IEEE, 2006. 285 – 288.

[9] Kamijo S, Matsushita Y, Ikeuchi K, et al. Traffic monitoring and accident detection at intersections [J]. IEEE Transactions on Intelligent Transportation System, 2000, 1(2): 108 – 118.

[10] Li N, Takaki S, Tomiokay Y, Kitazawa H. A multistage dataflow implementation of a deep convolutional neural network based on FPGA for high-speed object recognition [A]. 2016 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI)[C]. USA: IEEE, 2016. 165 – 168.

[11] Cadambi S, Majumdar A, Becchi M, Chakradhar S, Graf

H P. A programmable parallel accelerator for learning and classification[A]. Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques[C]. Austria:ACM,2010. 273 - 284.

[12] Chakradhar S, Sankaradas M, Jakkula V, Cadambi S. A dynamically configurable coprocessor for convolutional neural networks[J]. ACM SIGARCH Computer Architecture News,2010,38(3):247 - 257.

[13] Peemen M, Setio A A, Mesman B, Corporaal H. Memory-centric accelerator design for convolutional neural networks[A]. IEEE 31st International Conference on Computer Design (ICCD)[C]. USA:IEEE,2013. 13 - 19.

[14] Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton. ImageNet classification with deep convolutional neural network[J]. Communications of the ACM,2017,60(6):84 - 90.

[15] Zhang C, Li P, Sun G, et al. Optimizing FPGA-based accelerator design for deep convolution neural networks[A]. Proceedings of the 2015 ACM/SIGDA International Symposium on Field Programmable Gate Arrays[C]. USA:ACM,2015. 161 - 170.

[16] Huang C, Ni S, Chen G. A layer-based structured design of CNN on FPGA[A]. IEEE 12th International Conference on ASIC (ASICON)[C]. Guiyang: IEEE, 2017. 1037 - 1040.

作者简介

蹇　强　男,1993 年出生,甘肃陇南市礼县人. 浙江大学信息与电子工程学院硕士研究生,主要研究方向为片上系统.
E-mail:21631061@zju.edu.cn

张培勇　男,1977 年出生,安徽省安庆市人. 2004 年于浙江大学获博士生学位,同年就职于浙江大学超大规模集成电路研究所,主要从事超大规模集成电路设计.
E-mail:zhangpy@zju.edu.cn